Grounding Synchronous Deterministic Concurrency in Sequential Programming

Reinhard von Hanxleden
Insa Fuhrmann
Kiel University

Michael Mendler
Joaquín Aguado
Bamberg University

Grenoble, France
ETAPS-ESOP'14
April, 2014

Overview

A classical problem in *concurrent* programming.

Determinism and Dead-lock freedom in multi-thread shared-memory settings.

An approach for this.

Synchronous Programming (SP) has already solved this for reactive and embedded systems.

Sound generalisation of SP techniques for main stream programming.

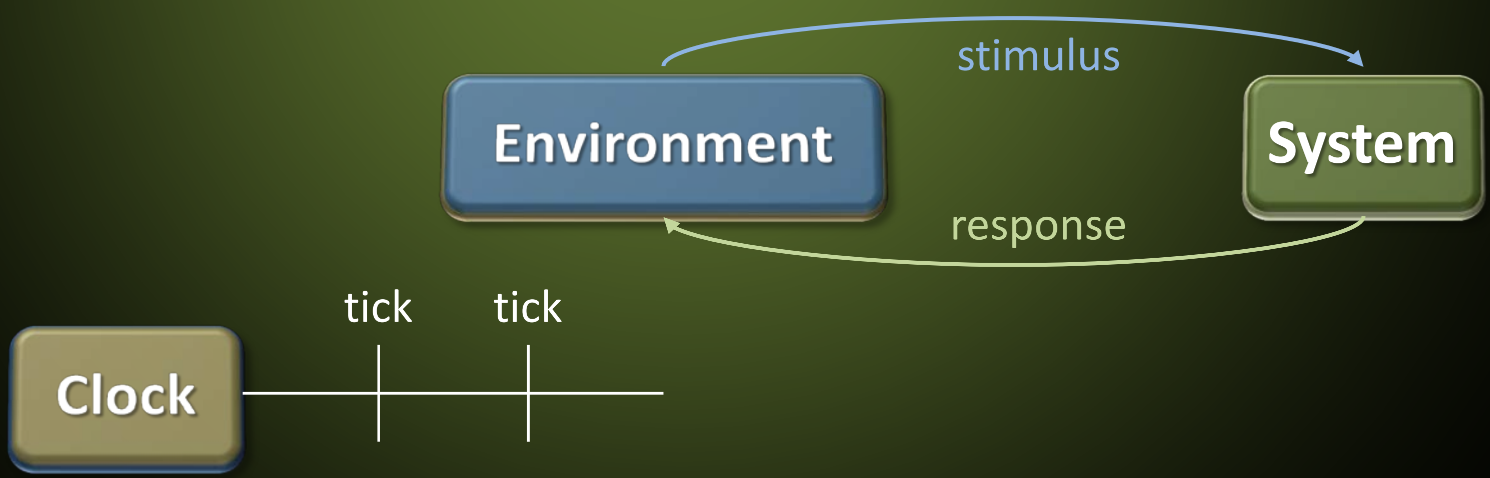
Context

Synchronous Model of Computation (SMoC):

Reactive and embedded systems.

Inspired in synchronous digital circuits.

Synchrony Hypothesis:

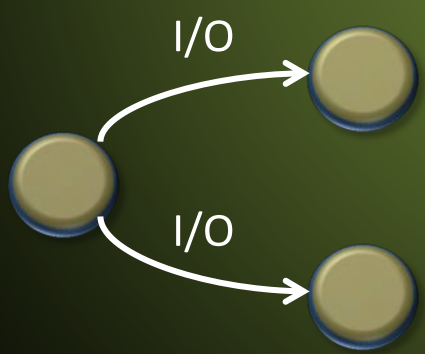


Context

Synchronous Model of Computation (SMoC):

Synchronisation is based on clocks and signals.

Classical view of computation: Mealy machine.

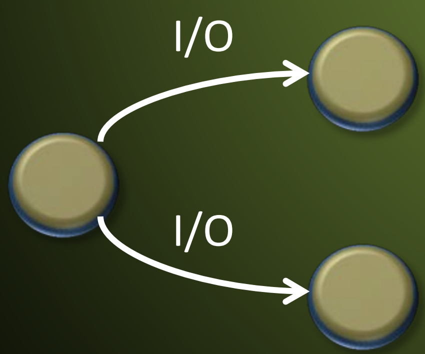


Context

Synchronous Model of Computation (SMoC):

This prevents deadlock and non-determinism.

The soundness of the automata model depends on the compiler verifying that the Synchrony Hypothesis is valid.



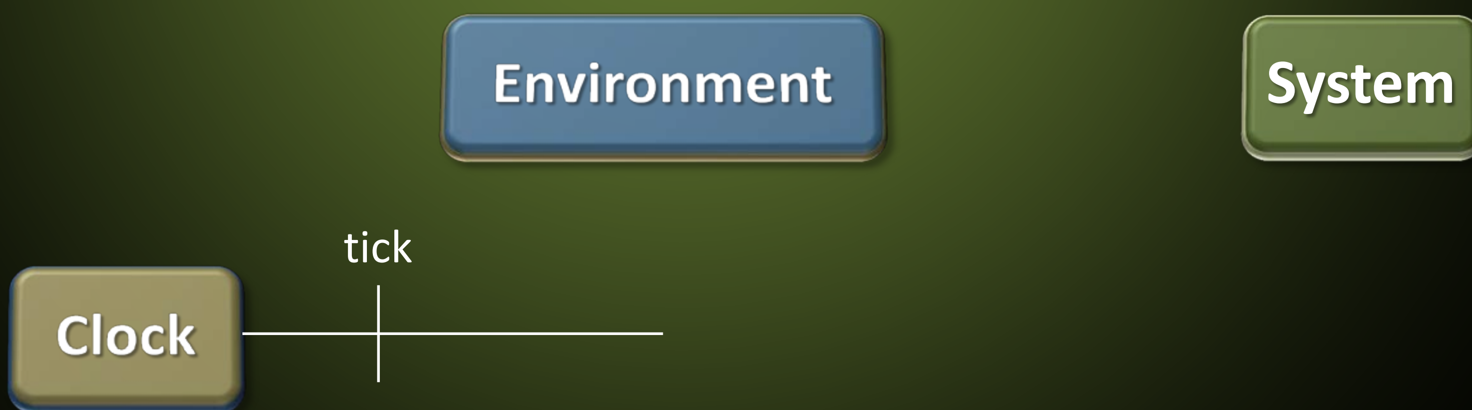
Thus, the synchronous interaction must satisfy stringent causality requirements.

Context

Synchronous Model of Computation (SMoC):

Yet, the Synchrony Hypothesis is not compositional !

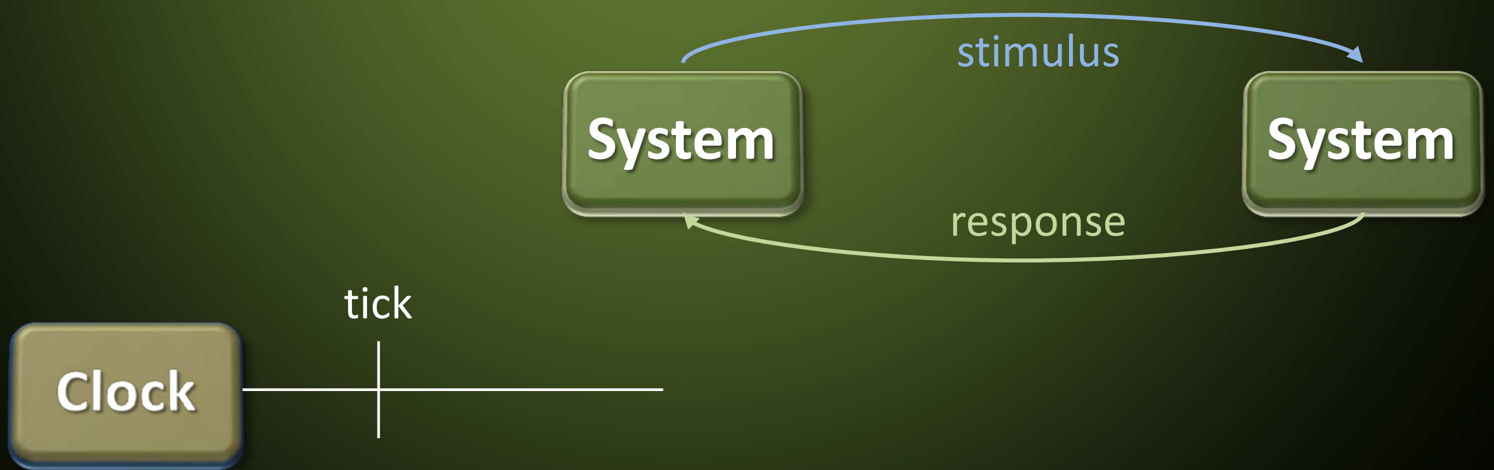
This is aggravated by the fact that reaction to absence is allowed in some SMoC languages.



Context

Synchronous Model of Computation (SMoC):

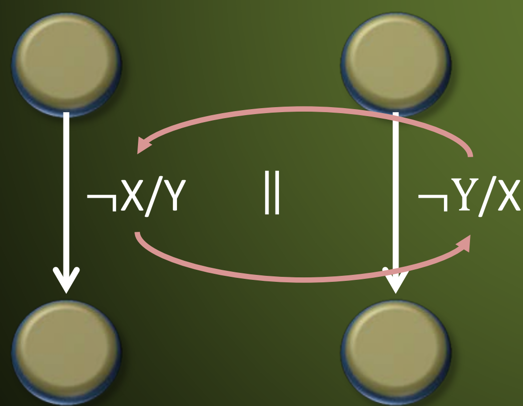
We cannot (compositionally) understand *Mealy* machine abstraction without causality analysis in micro-steps.

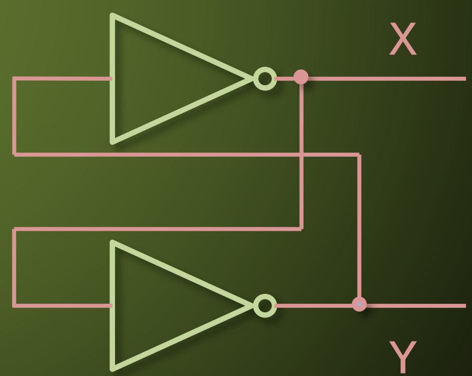


Context

Synchronous Model of Computation (SMoC):

We cannot (compositionally) understand *Mealy* machine abstraction without causality analysis in micro-steps.





Circuit

$X = \neg Y$

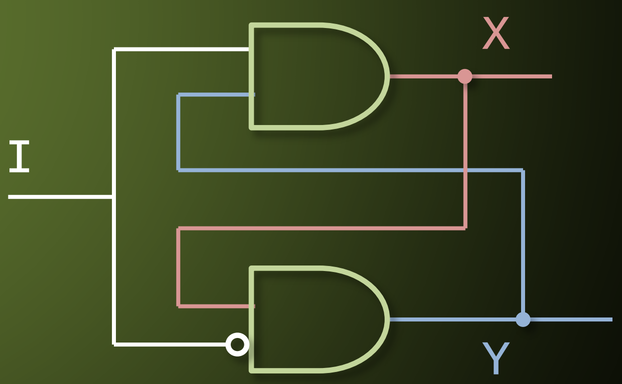
$Y = \neg X$

Context

Esterel **synchronous language (**Gérard Berry**):**

Constructiveness relates to electrical stabilisation.
Cyclic circuit that **stabilises** for all (non-inertial) delays.

```
present I then
   present X then emit Y end
else
   present Y then emit X end
end;
pause;
...
```



Circuit

```
X = I ∧ Y
Y = ¬I ∧ X
```

Context

Synchronous Model of Computation (SMoC):

Gérard Berry (*Esterel*) has solved this in the context of synchronous digital circuits.

Causality analysis establishes consistency of a synchronous macro-step with respect to an asynchronous micro-step execution model.

What does this mean for shared-memory multi-threaded code?

Contributions

Esterel is extended (first time) as follows:

For multi-threaded shared-memory programs:

Two notions of Berry-constructiveness (Δ_0, Δ_1):

Δ_0 permits explicit initialisations.

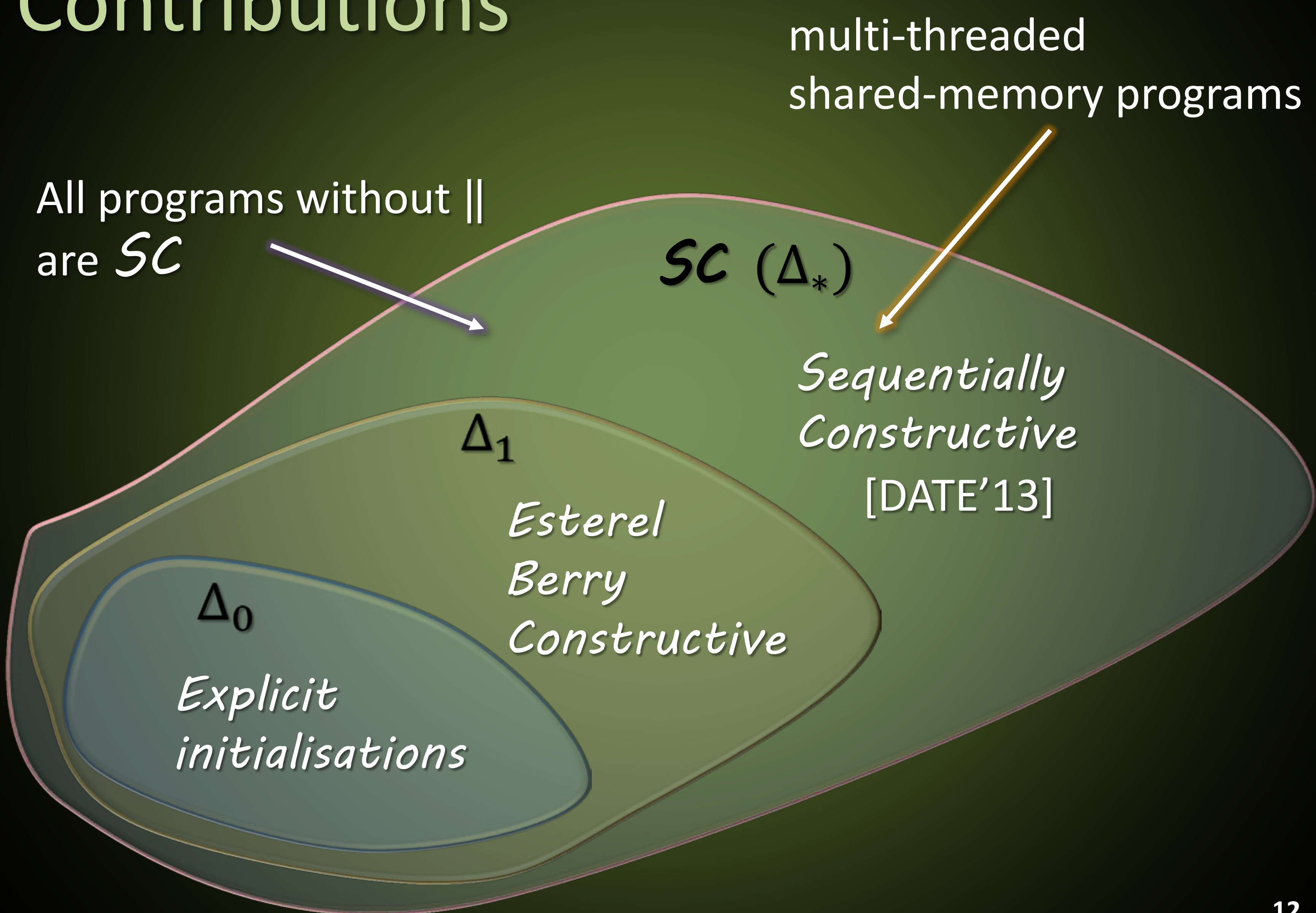
Δ_1 corresponds to *Esterel*.

These are presented as fixed point analyses in abstract domains of variable statuses:

Novel characterisation of must-cannot.

Formally, constructive semantics of *Esterel* generalises to *SC*.

Contributions



Language

The syntax (finite tick behaviour) is given by the BNF:

$$P \coloneqq \epsilon \mid ¡s \mid !s \mid s?\,P : P \mid P \| P \mid P ; P$$

This contains the necessary control structures for capturing multiple variable accesses as they occur inside macro-steps.

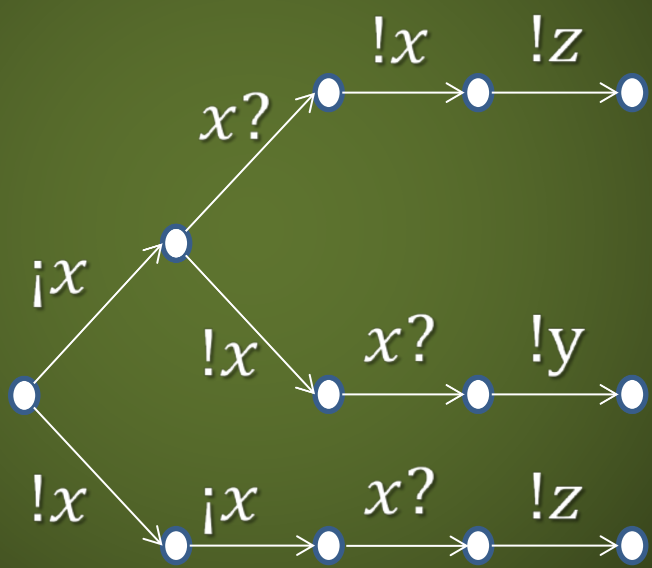
Programs manipulate Boolean variables $\mathrm{B} = \{1,0\}$ that emulate the synchronous signal statuses:

present $(1, True)$
absent $(0, False)$

Operational Semantics

Concurrent control flow is *descriptive*.
Sequential control flow is *prescriptive*.

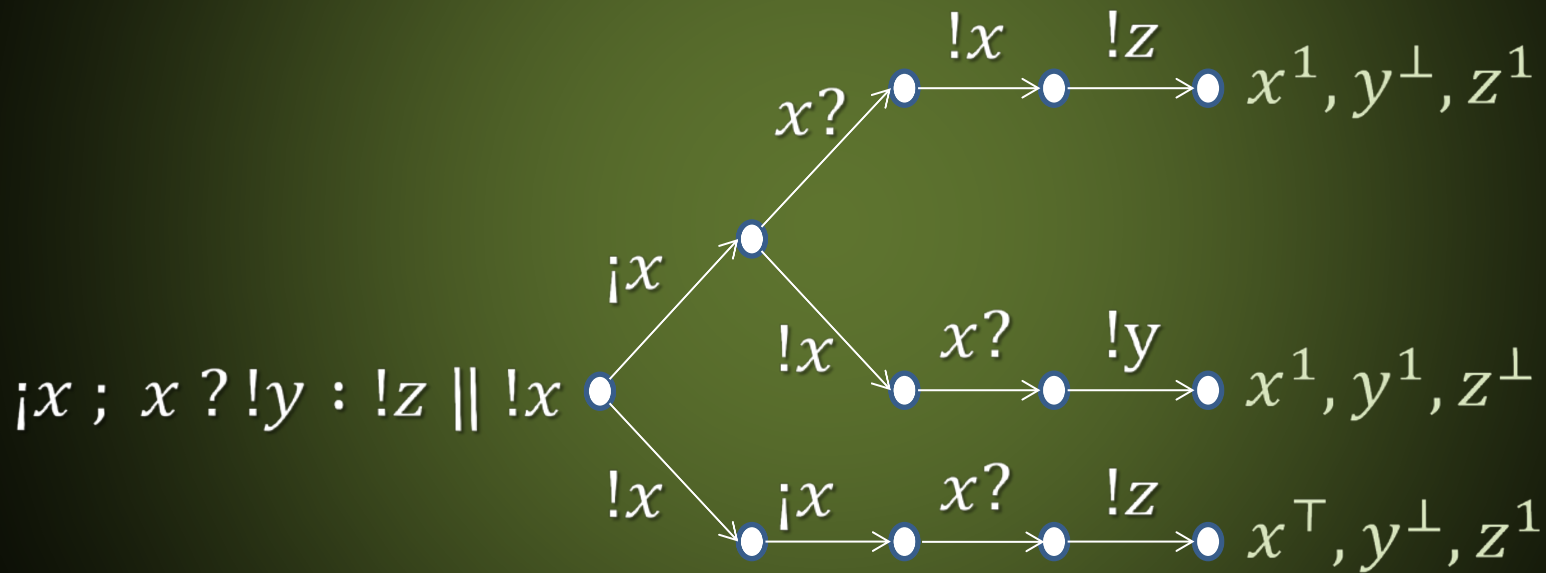
¡x ; x ? !y : !z || !x



Abstract Value Domain

The behaviour off a variable takes place in a 4-value domain:

$D = \{\bot < 0 < 1 < \top\}$

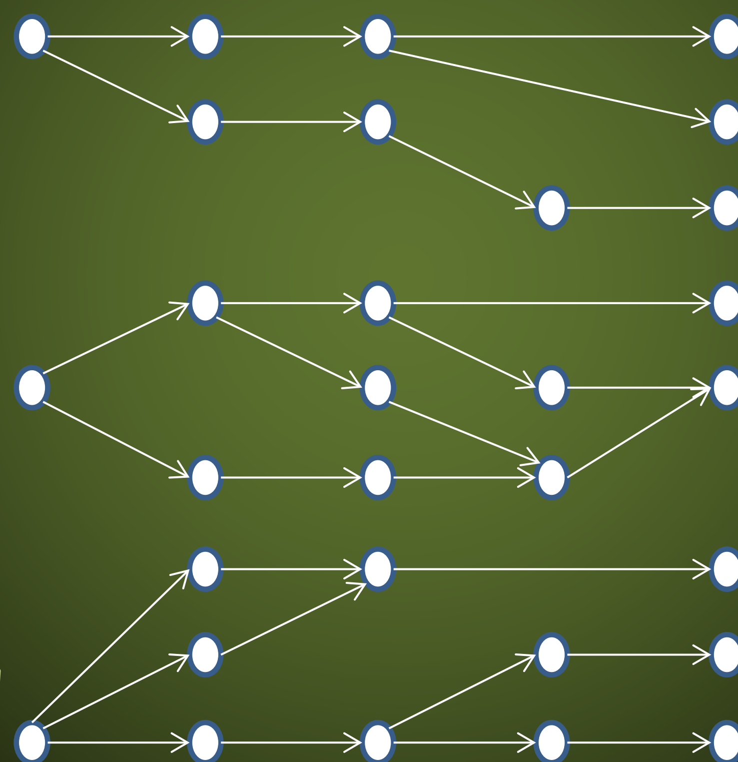


IUR Protocol requires ⊤

Abstract Value Domain

Closed intervals.

Arbitrary initial memory



x	y	z
0	1	$\bot$
$\bot$	1	$\bot$
0	$\top$	1
0	0	0
$\bot$	1	0
$\bot$	$\top$	1
0	1	$\bot$
0	1	1

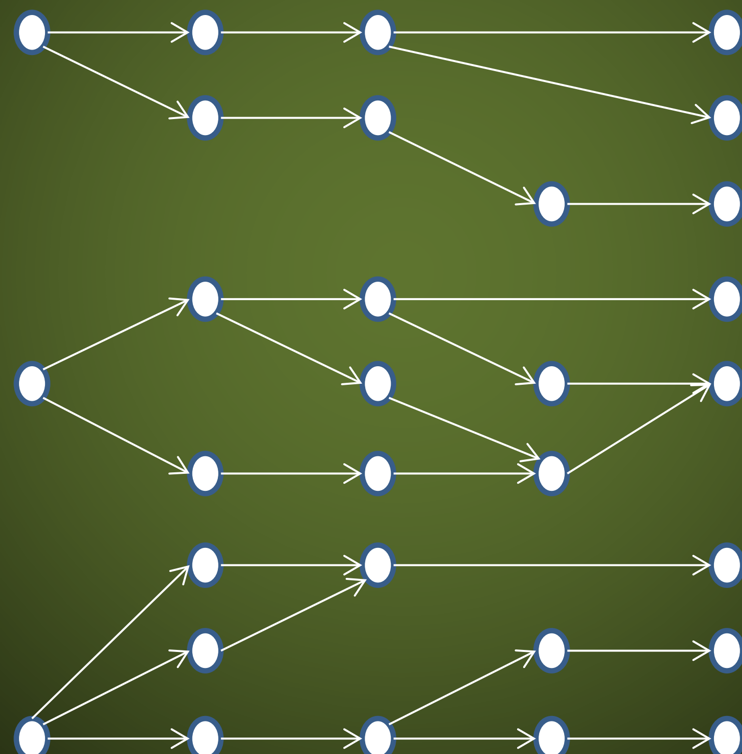
Abstract Value Domain

Closed intervals.

$x : [\bot, \top]$

$y : [\bot, \top]$

$z : [\bot, \top]$



x	y	z
0	1	$\bot$
$\bot$	1	$\bot$
0	$\top$	1
0	0	0
$\bot$	1	0
$\bot$	$\top$	1
0	1	$\bot$
0	1	1

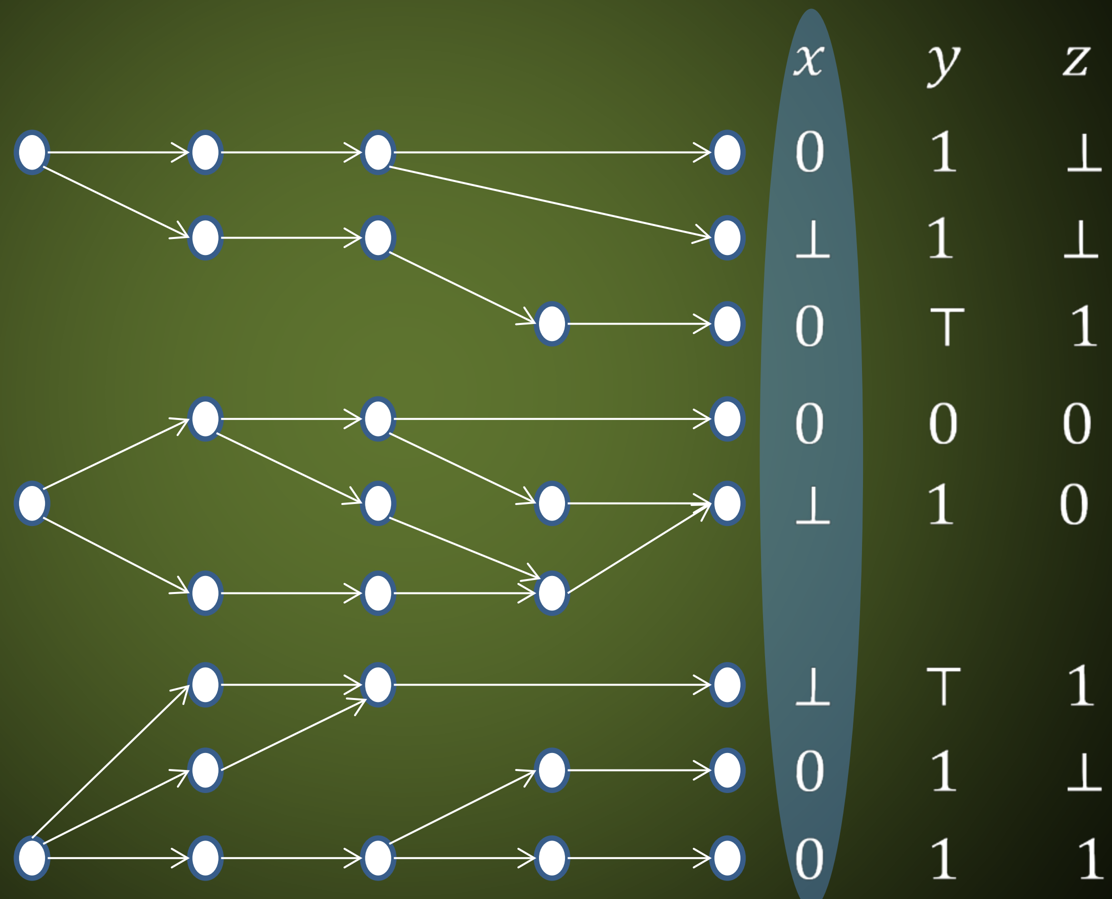
Abstract Value Domain

Closed intervals.

$x : [\bot, 0]$

$y : [\bot, \top]$

$z : [\bot, \top]$



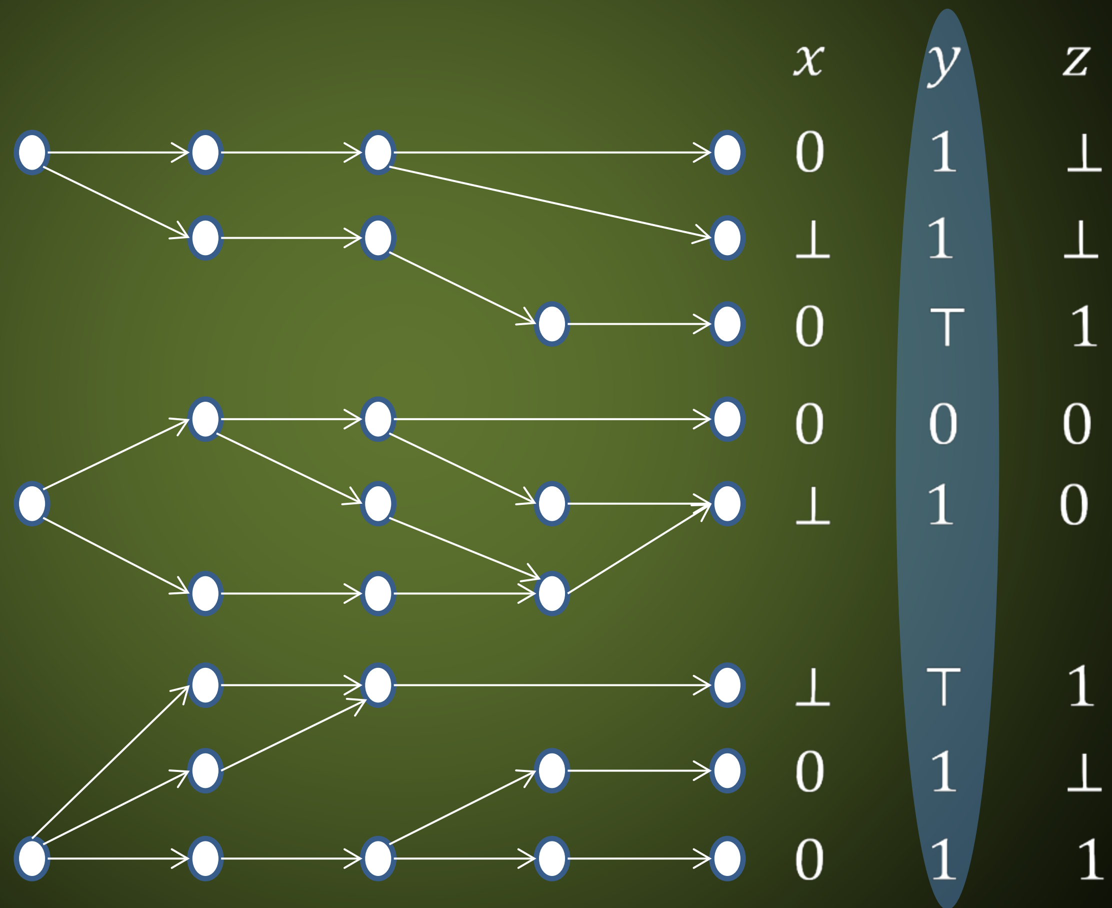
Abstract Value Domain

Closed intervals.

$x : [\bot, 0]$

$y : [0, \top]$

$z : [\bot, \top]$



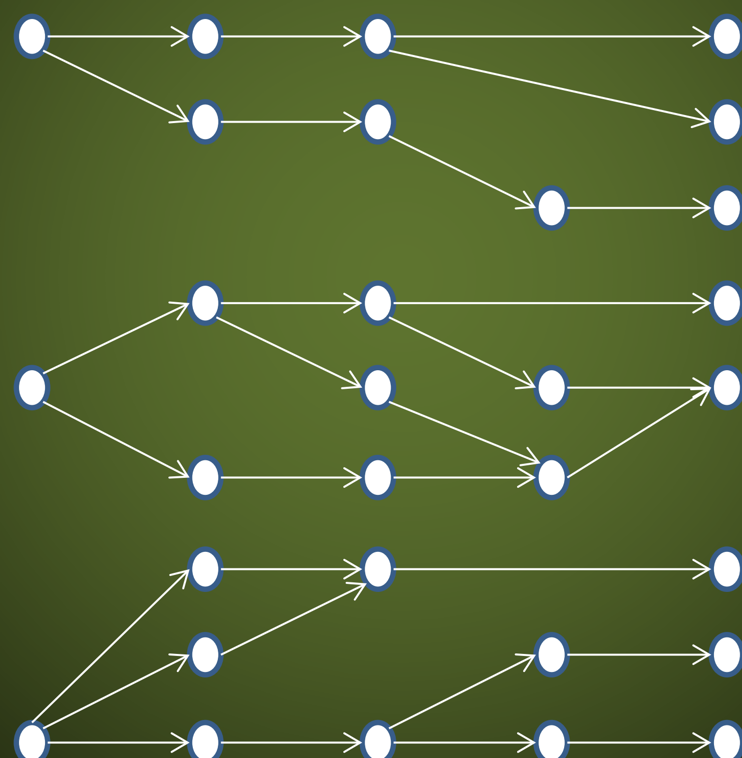
Abstract Value Domain

Closed intervals.

$x : [\bot, 0]$

$y : [0, \top]$

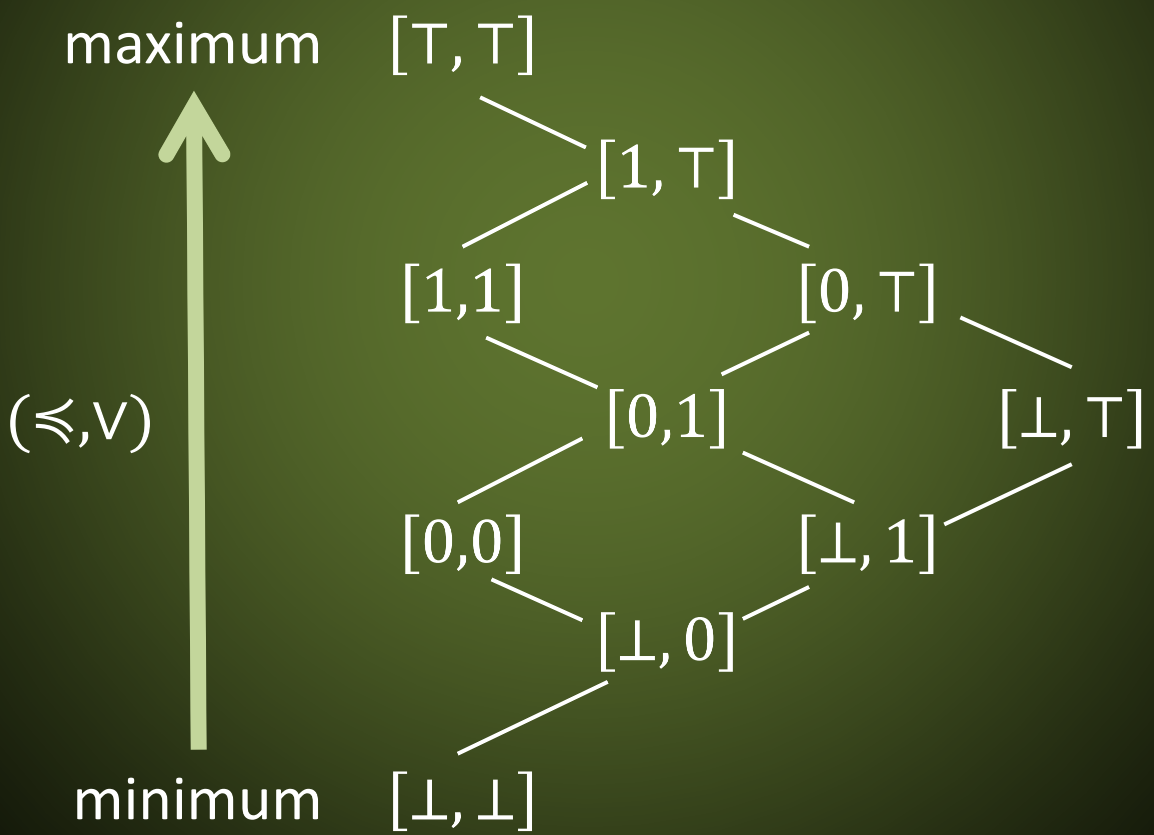
$z : [\bot, 1]$



x	y	z
0	1	$\bot$
$\bot$	1	$\bot$
0	$\top$	1
0	0	0
$\bot$	1	0
$\bot$	$\top$	1
0	1	$\bot$
0	1	1

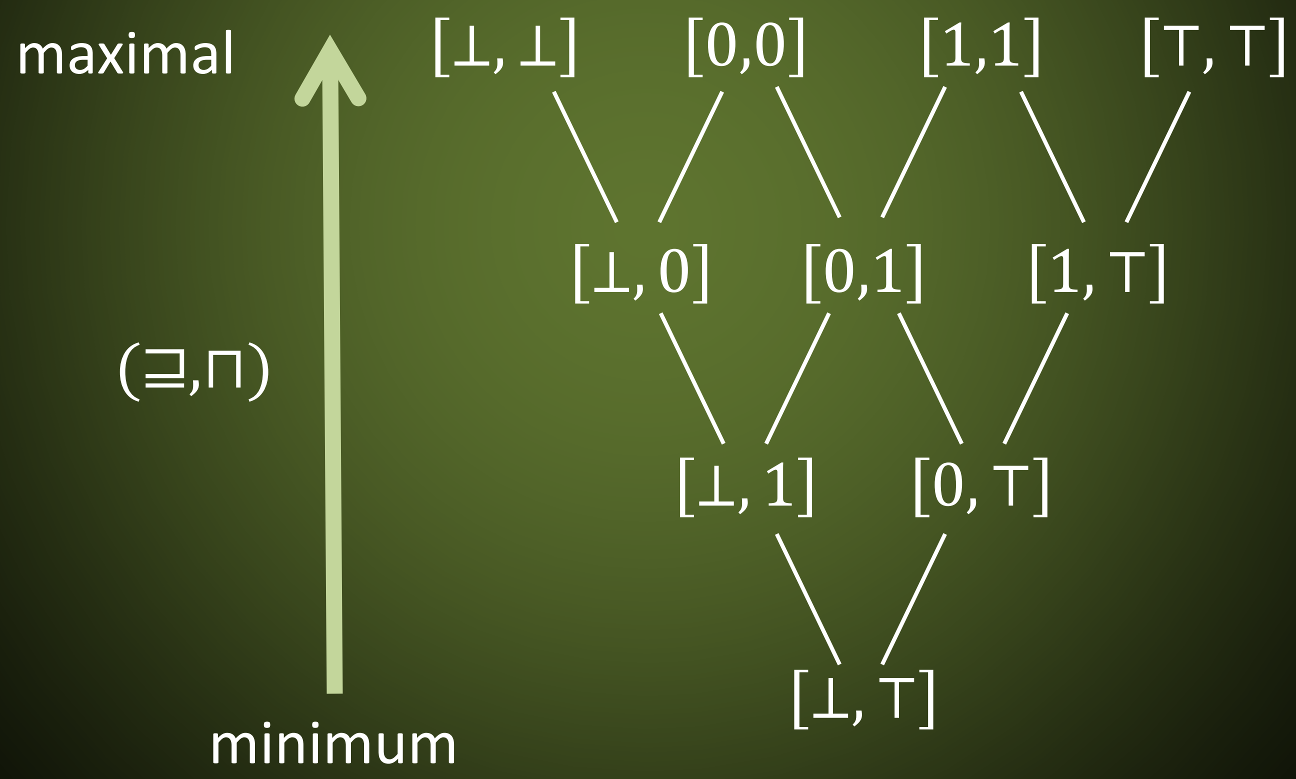
Abstract Value Domain

Point-wise (sequential) ≼-lattice:



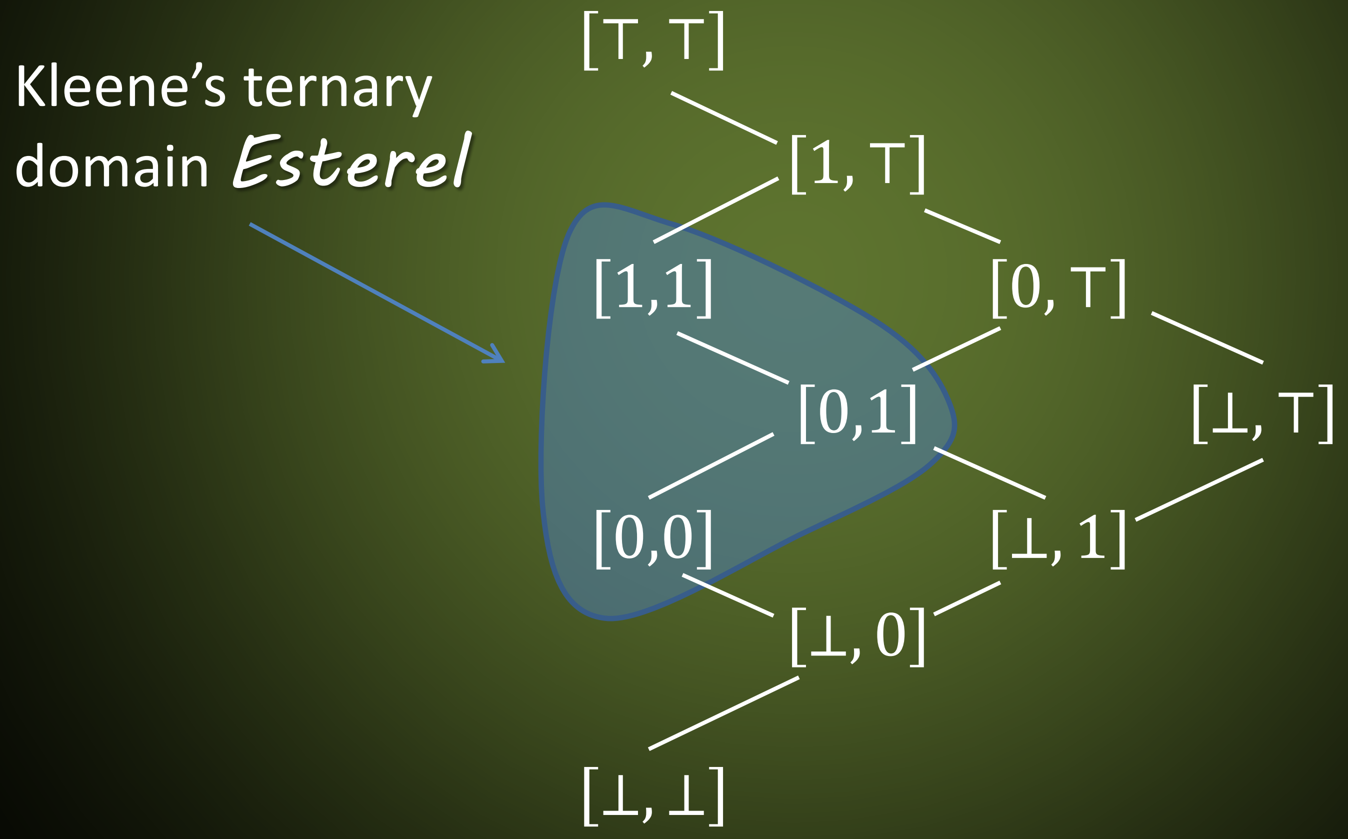
Abstract Value Domain

Information (concurrent) ⊑-semi-lattice:



Abstract Value Domain

Statuses of variables are keep in *environments* $E : V \mapsto I(D)$.



Abstract Analysis

Sequential-Concurrent Reaction Model

Sequential Environment

Initialisation under which P is activated.

$\langle\langle P \rangle\rangle_C^S$

Value of variables sequentially before P is started.

Concurrent Environment

External stimulus which is concurrent with P.

Abstract Analysis

Sequential-Concurrent Reaction Model

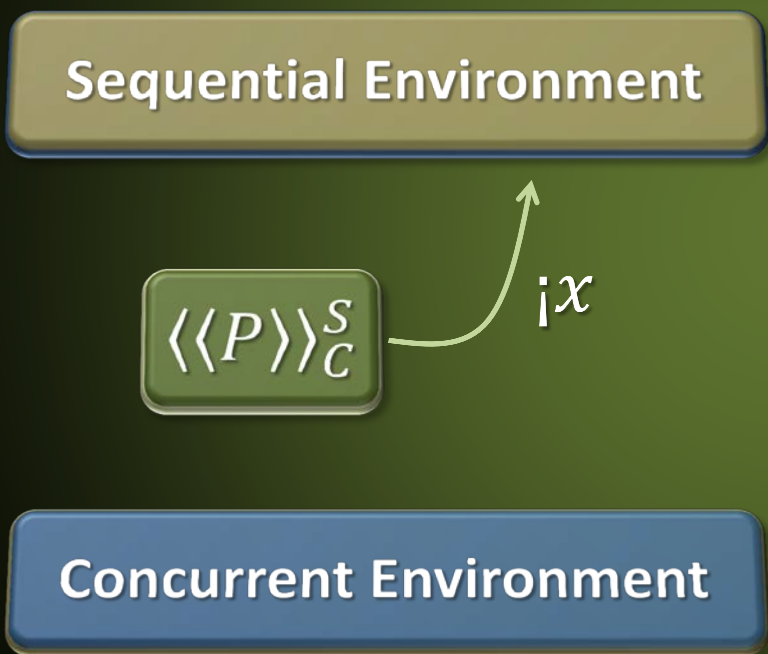
Sequential Environment $[\bot, \bot]$

$\langle\langle P \rangle\rangle_C^S$

Concurrent Environment $[\bot, \top]$

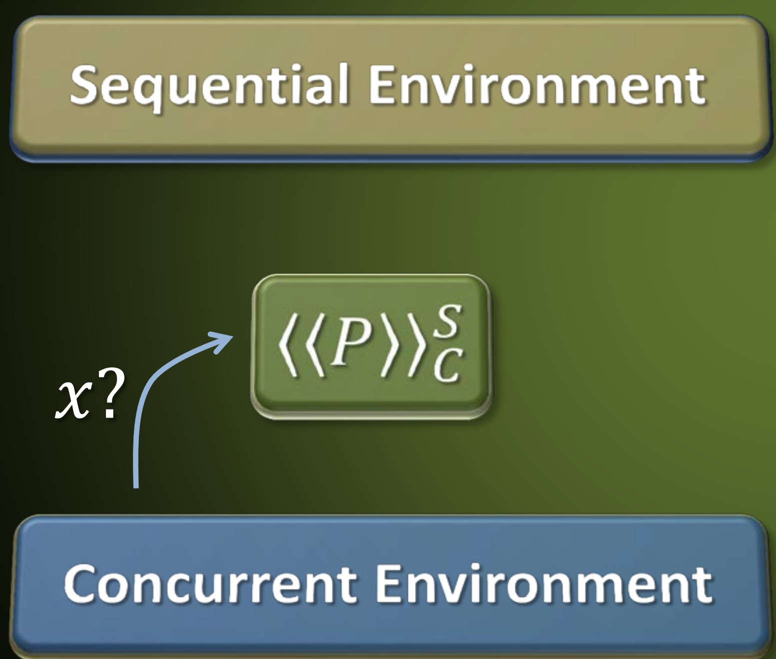
Abstract Analysis

Sequential-Concurrent Reaction Model



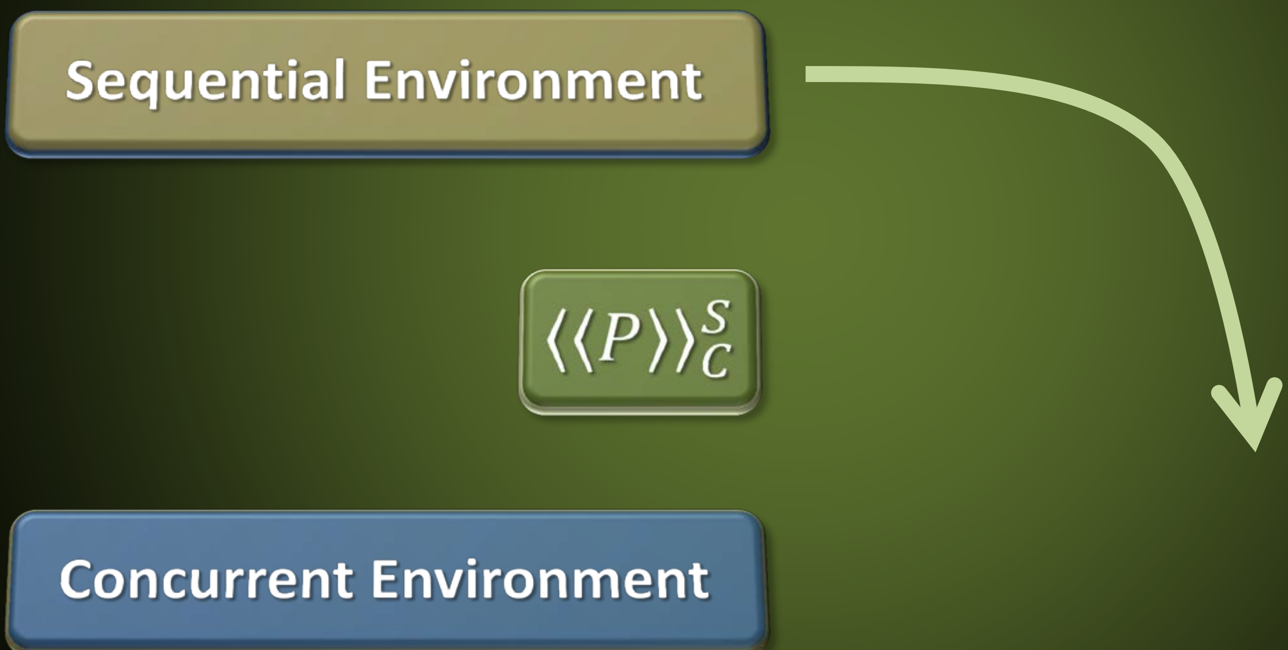
Abstract Analysis

Sequential-Concurrent Reaction Model



Abstract Analysis

Sequential-Concurrent Reaction Model



Abstract Analysis

Sequential-Concurrent Reaction Model

$\langle\langle P \rangle\rangle_C^S$

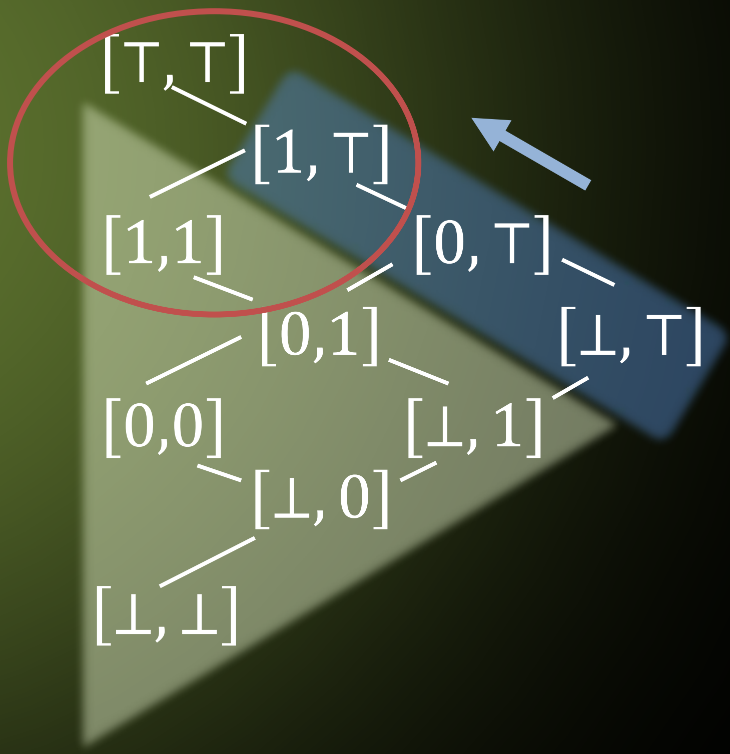
Concurrent Environment $\sqcap$ **Sequential Environment**

Abstract Analysis

The denotational semantics is given by a *Response Function* that determines constructive (non-speculative) information on the instantaneous response of a program.

$\langle\langle \epsilon \rangle\rangle_C^S := S$

$\langle\langle !x \rangle\rangle_C^S := S \vee \{\langle x^1 \rangle\}$



Abstract Analysis

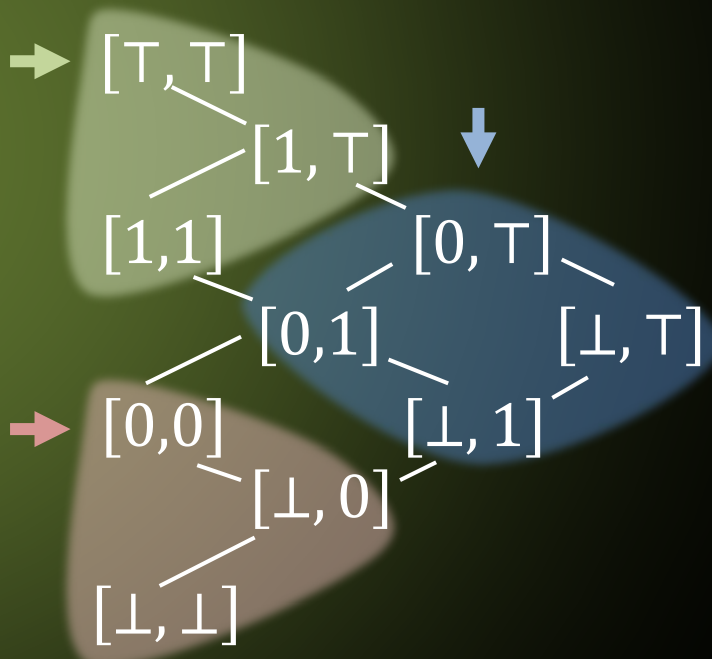
The denotational semantics is given by a *Response Function* that determines constructive (non-speculative) information on the instantaneous response of a program.

$$\langle\langle \epsilon \rangle\rangle_C^S := S$$

$$\langle\langle !x \rangle\rangle_C^S := S \vee \{\langle x^1 \rangle\}$$

$$\langle\langle ¡x \rangle\rangle_C^S := \begin{cases} S \vee \{\langle x^\top \rangle\} & \text{if } 1 \preccurlyeq S(x) \\ S \vee \{\langle x^0 \rangle\} & \text{if } S(x) \preccurlyeq 0 \\ S \vee \{\langle x^{[0,\top]} \rangle\} & \text{otherwise} \end{cases}$$

$$\langle\langle P \| Q \rangle\rangle_C^S := \langle\langle P \rangle\rangle_C^S \vee \langle\langle Q \rangle\rangle_C^S$$



Abstract Analysis

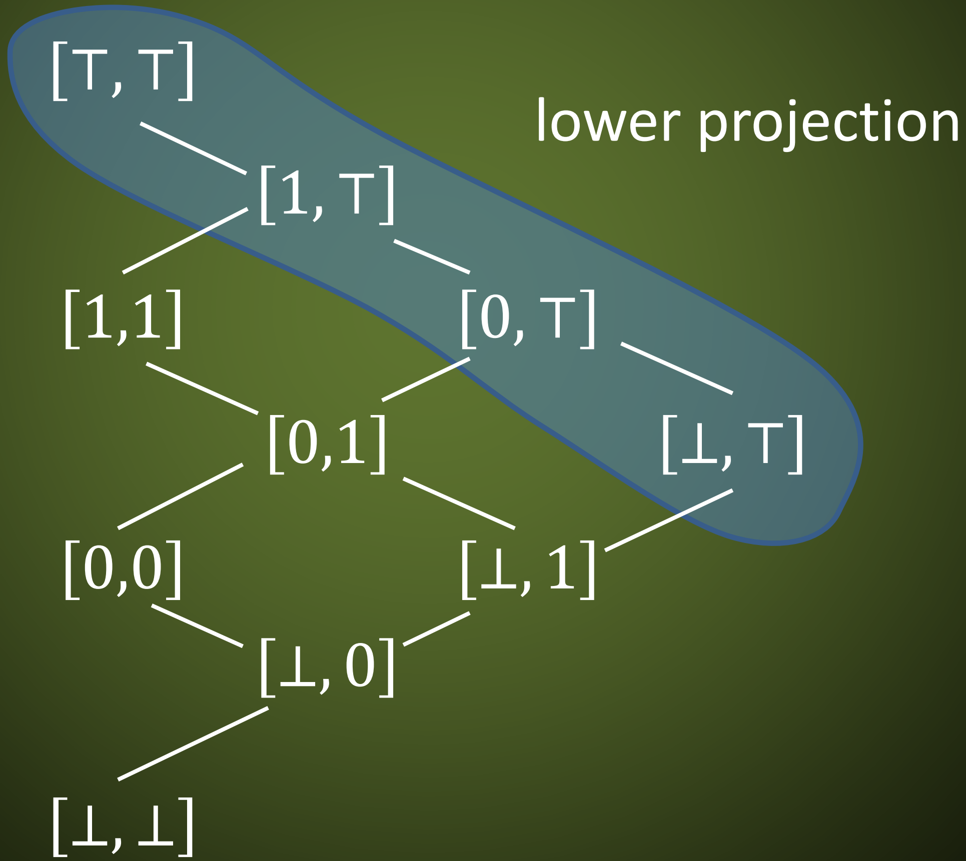
The denotational semantics is given by a Response Function.

$$\langle\langle x \mathbin{?} P : Q \rangle\rangle_C^S \coloneqq \begin{cases} \langle\langle P \rangle\rangle_C^S & \text{if } x^1 \in C \\ \langle\langle Q \rangle\rangle_C^S & \text{if } x^0 \in C \\ S \vee upp(\langle\langle P \rangle\rangle_C^S) \vee upp(\langle\langle Q \rangle\rangle_C^S) & \text{otherwise} \end{cases}$$

$$\langle\langle P ; Q \rangle\rangle_C^S \coloneqq \begin{cases} \langle\langle Q \rangle\rangle_C^{\langle\langle P \rangle\rangle_C^S} & \text{if } cmpl(P, C) = \{0\} \\ P \vee upp\left(\langle\langle Q \rangle\rangle_C^{\langle\langle P \rangle\rangle_C^S}\right) & \text{otherwise} \end{cases}$$

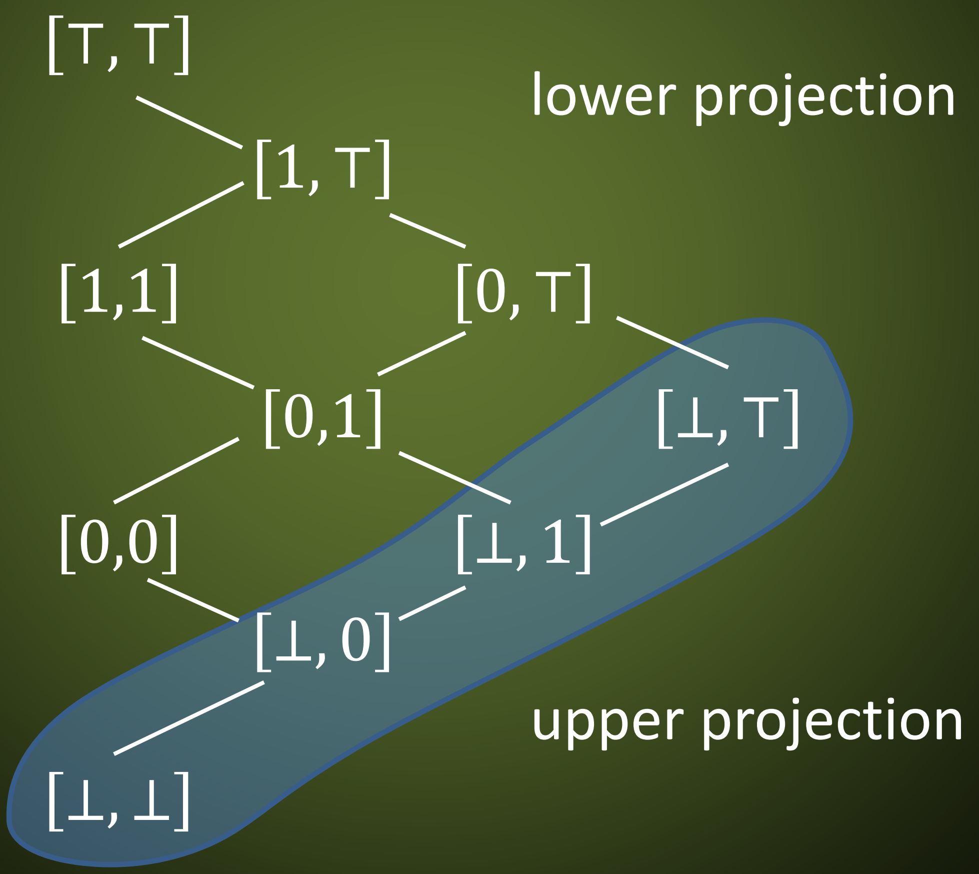
Abstract Analysis

Statuses of variables are keep in *environments* $E : V \mapsto I(D)$.



Abstract Analysis

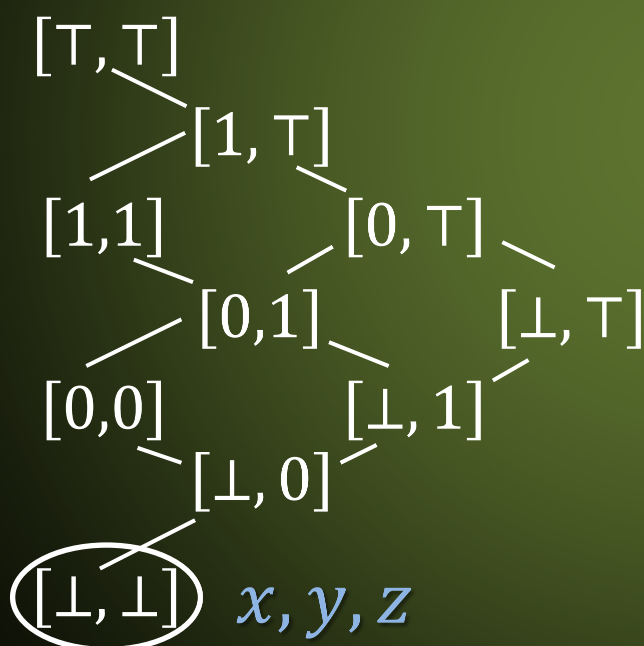
Statuses of variables are keep in *environments* $E : V \mapsto I(D)$.



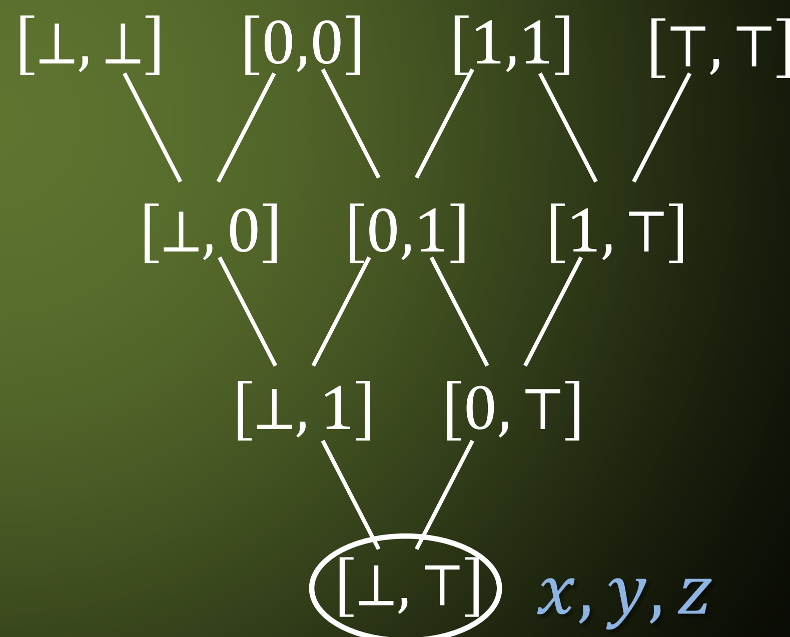
Abstract Analysis

Example

$$\langle\langle \text{¡}x\ ;\ x\ ?\ !y : !z \parallel !x \rangle\rangle_C^S$$



Sequential

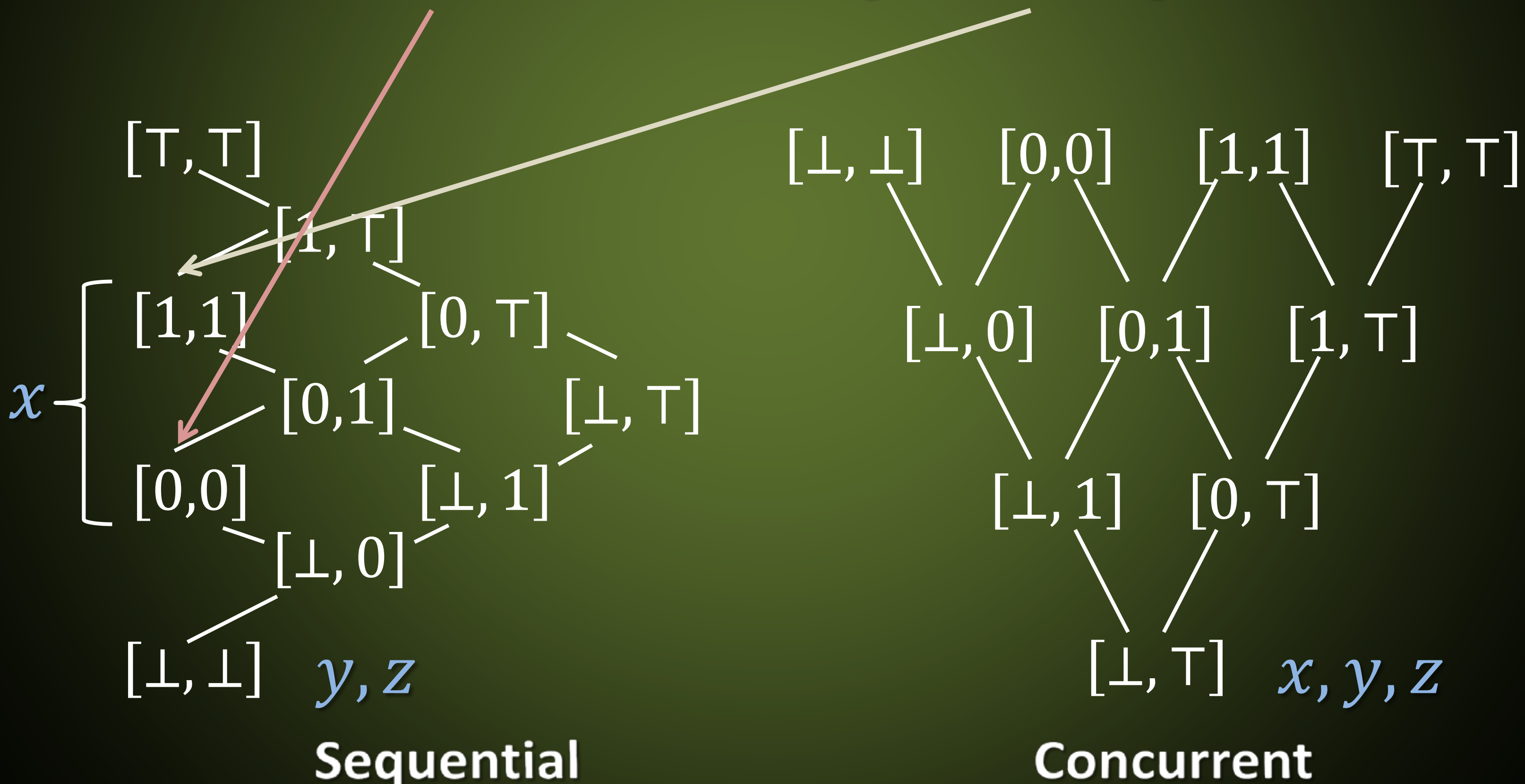


Concurrent

Abstract Analysis

Example

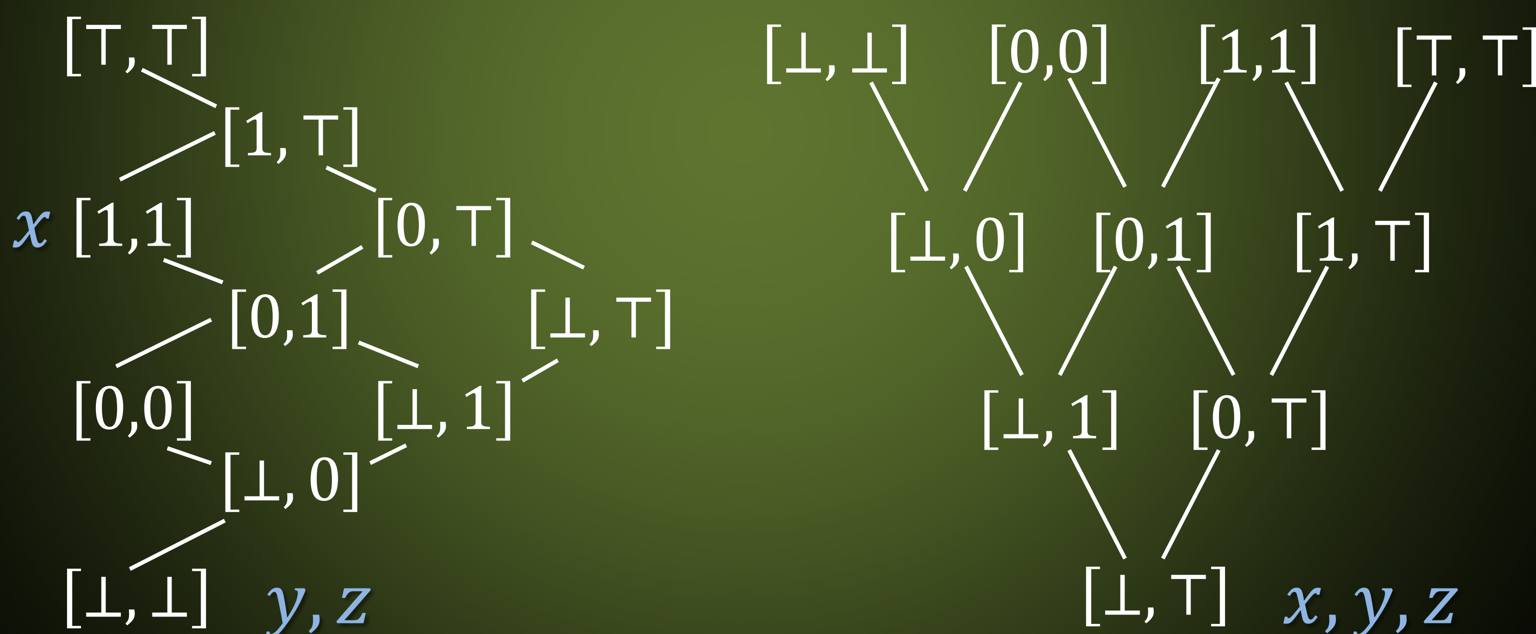
$$\langle\langle ¡x\ ;\ x\ ?\ !y : !z \rangle\rangle_C^S \vee \langle\langle !x \rangle\rangle_C^S$$



Abstract Analysis

Example

$$\langle\langle ¡x \;;\; x \;?\; !y : !z \rangle\rangle_C^S \vee \langle\langle !x \rangle\rangle_C^S$$

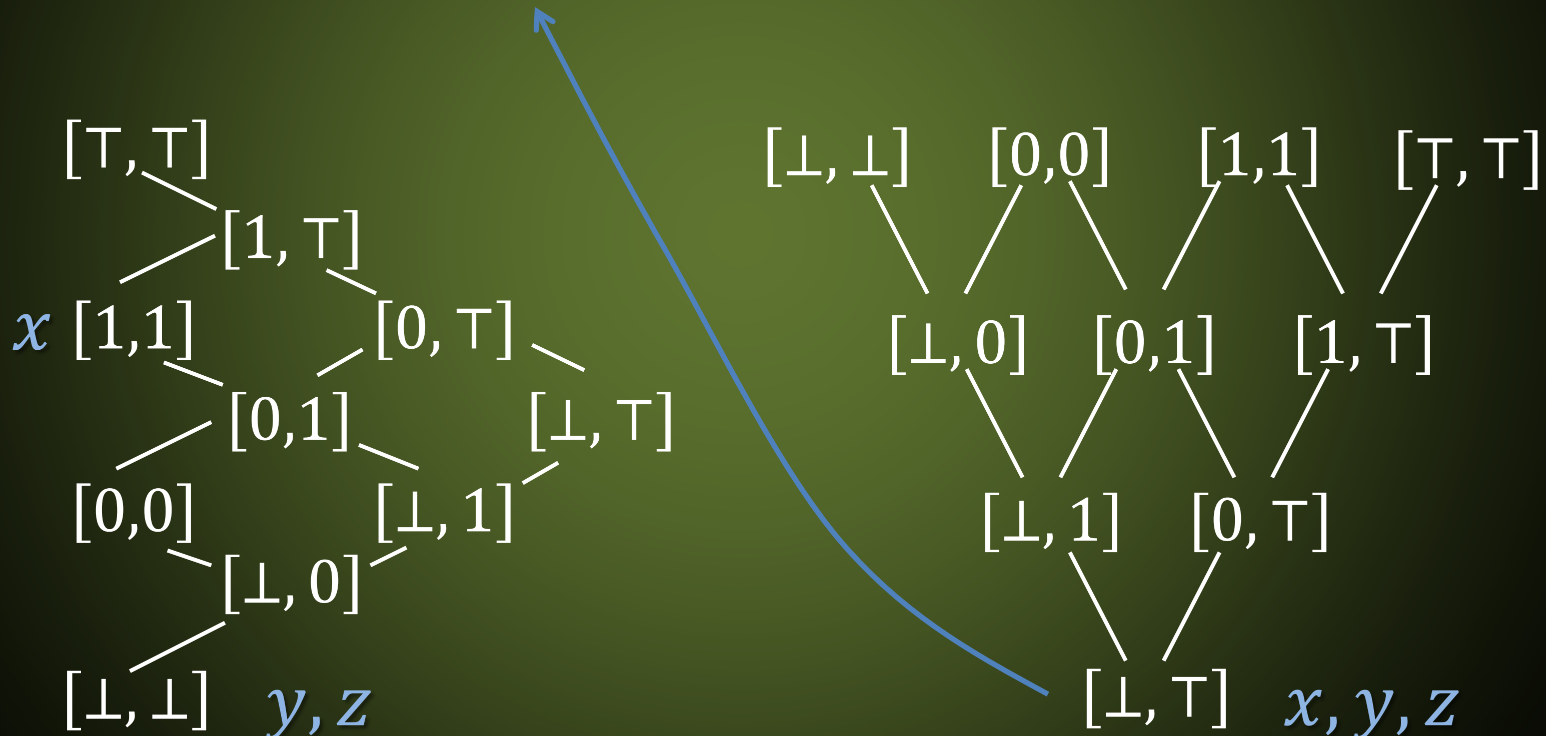


Sequential **Concurrent**

Abstract Analysis

Example

$$\langle\langle ¡x\ ;\ x\ ?\ !y : !z \rangle\rangle_C^S \vee \langle\langle !x \rangle\rangle_C^S$$

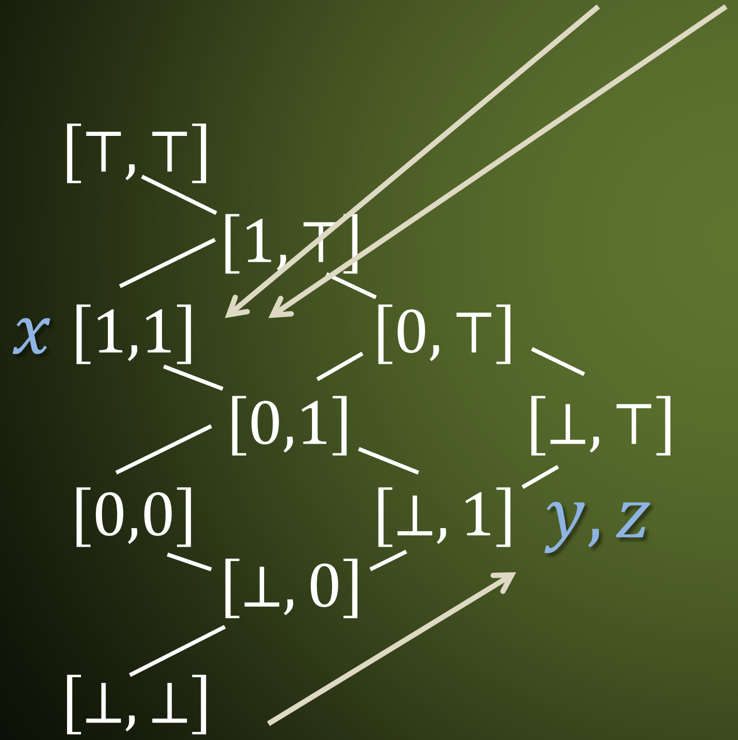


Sequential **Concurrent**

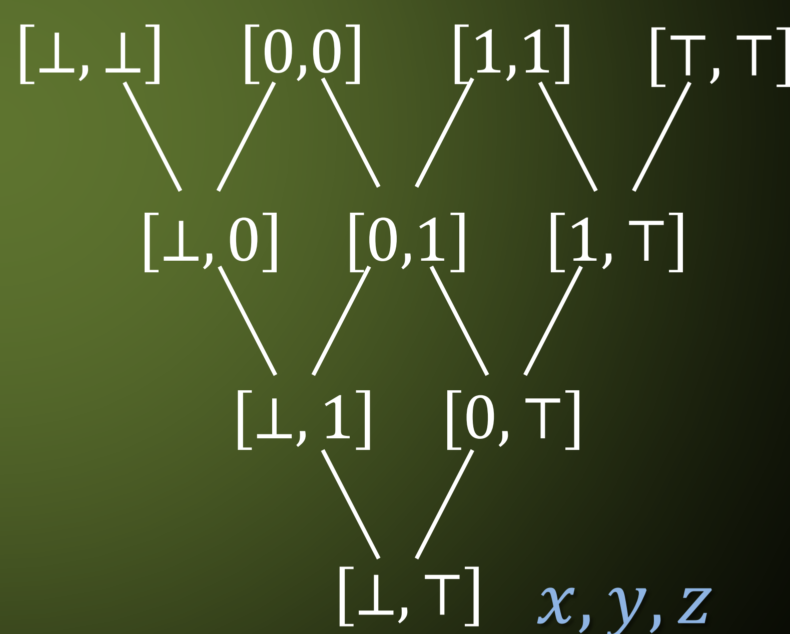
Abstract Analysis

Example

$$\langle\langle ¡x \; ; \; x \; ? \; !y \; : \; !z \rangle\rangle_C^S \vee \langle\langle !x \rangle\rangle_C^S$$



Sequential

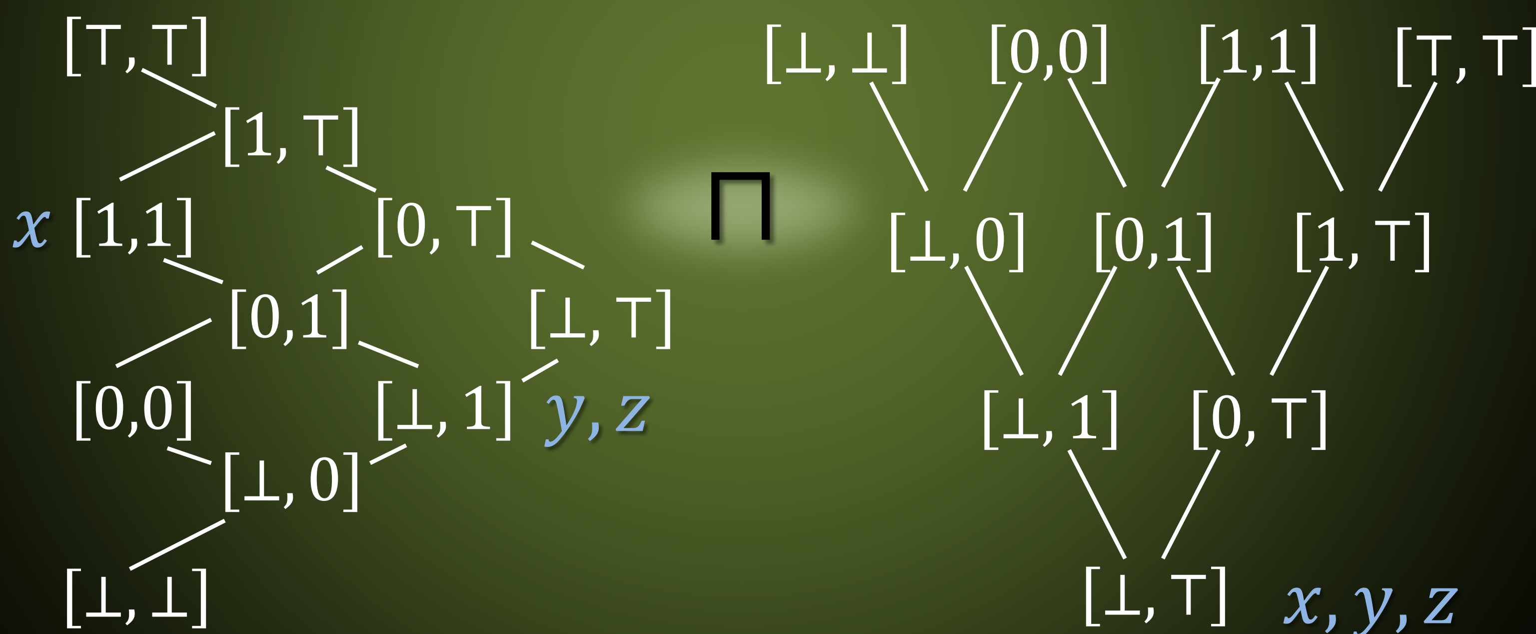


Concurrent

Abstract Analysis

Example

$$\langle\langle ¡x \;;\; x\,?\,!y : !z \parallel !x \rangle\rangle_C^S$$

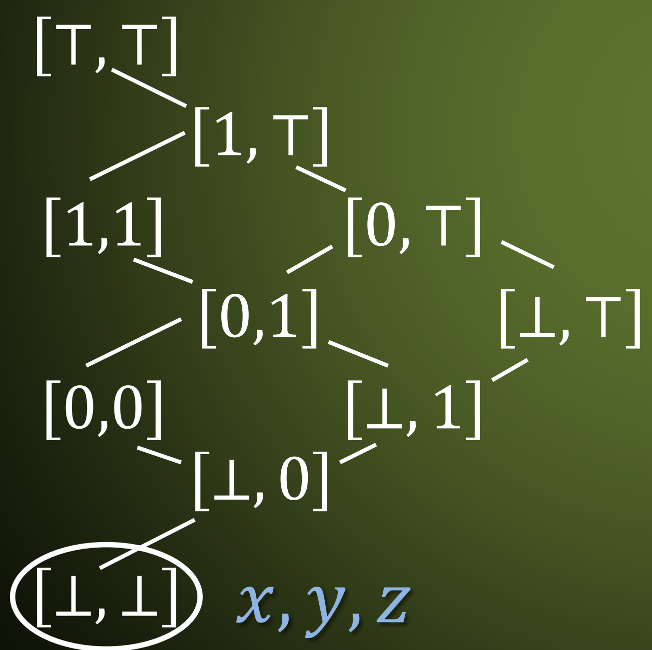


Sequential **Concurrent**

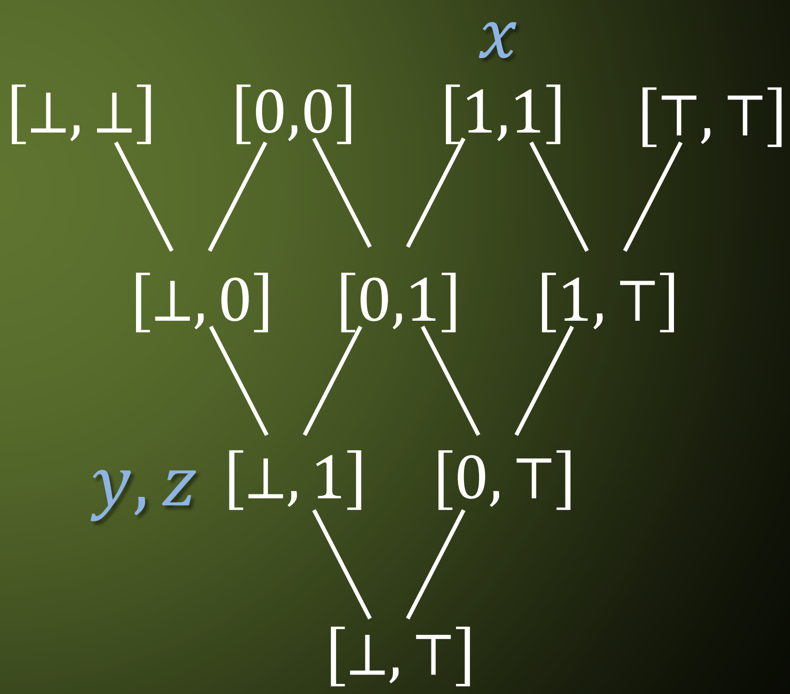
Abstract Analysis

Example

$$\langle\langle ¡x \; ; \; x \; ? \; !y : !z \parallel !x \rangle\rangle_C^S$$



Sequential

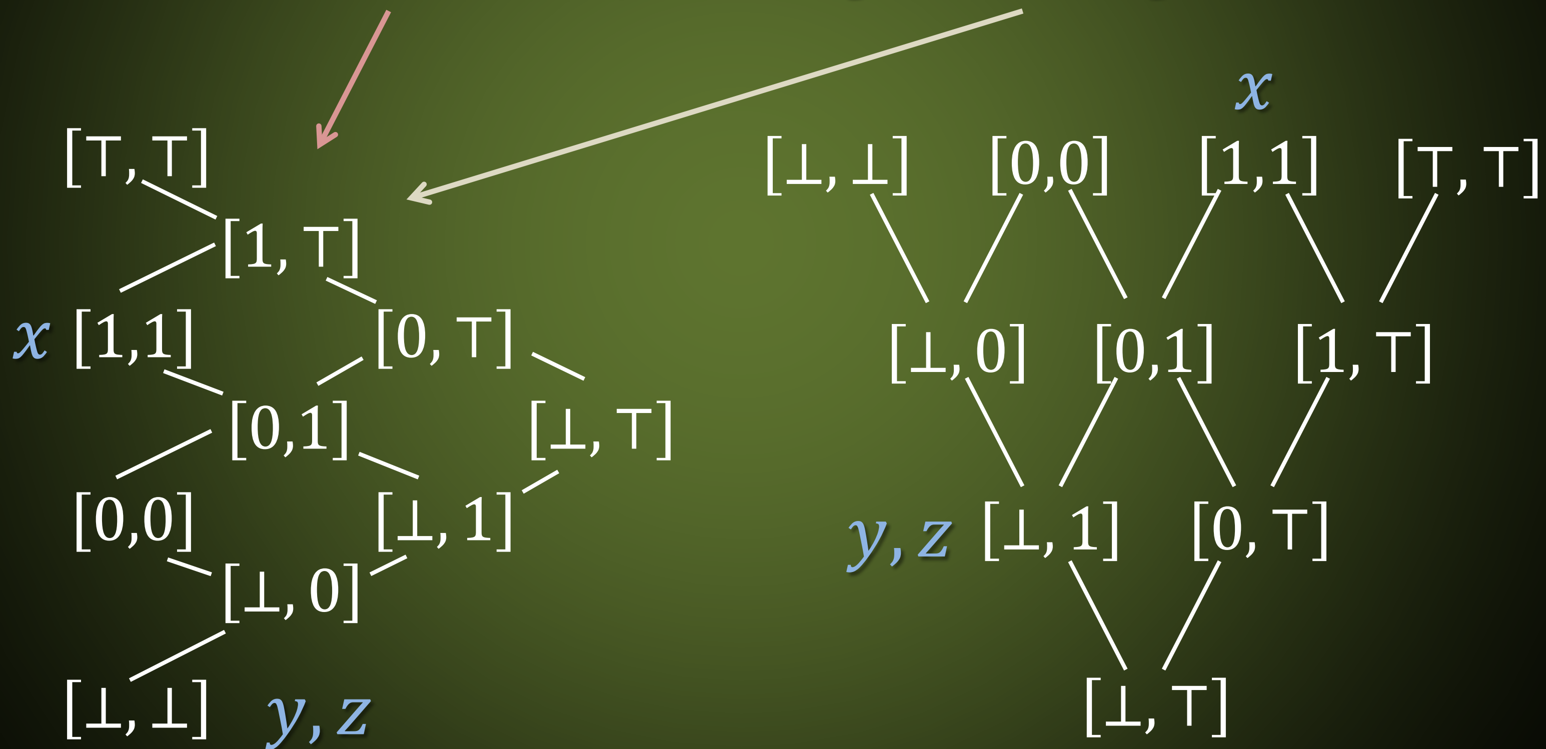


Concurrent

Abstract Analysis

Example

$$\langle\langle ¡x \; ; \; x \; ? \; !y : !z \rangle\rangle_C^S \vee \langle\langle !x \rangle\rangle_C^S$$

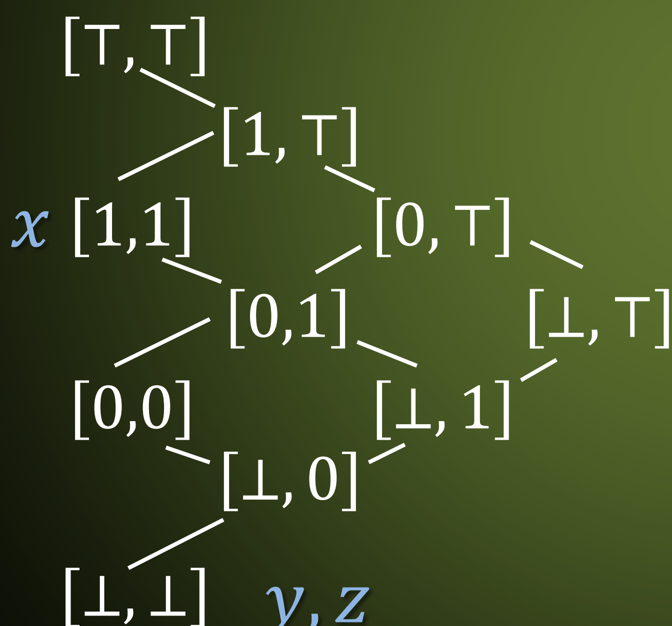


Sequential **Concurrent**

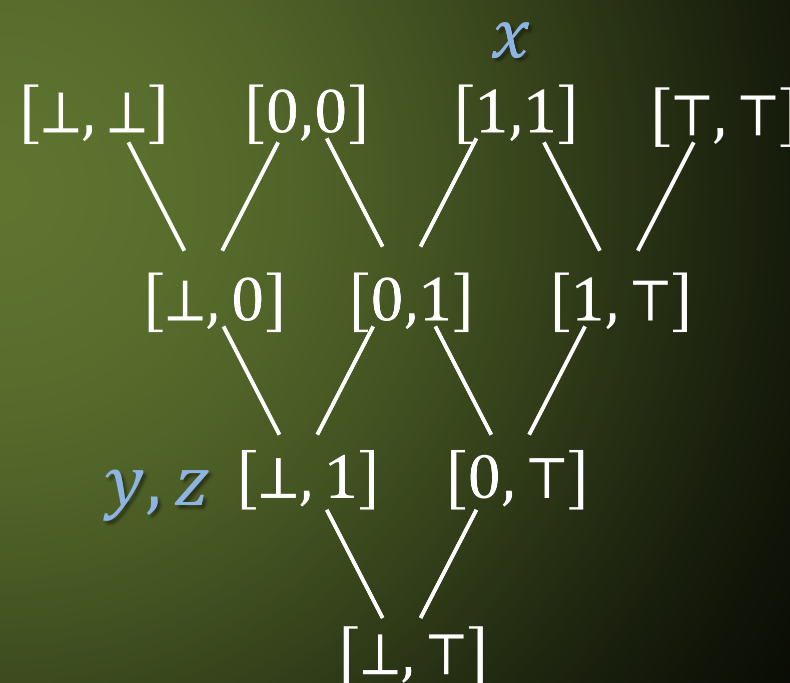
Abstract Analysis

Example

$$\langle\langle ¡x \;;\; x \;?\; !y : !z \rangle\rangle^{S}_{C} \vee \langle\langle !x \rangle\rangle^{S}_{C}$$



Sequential

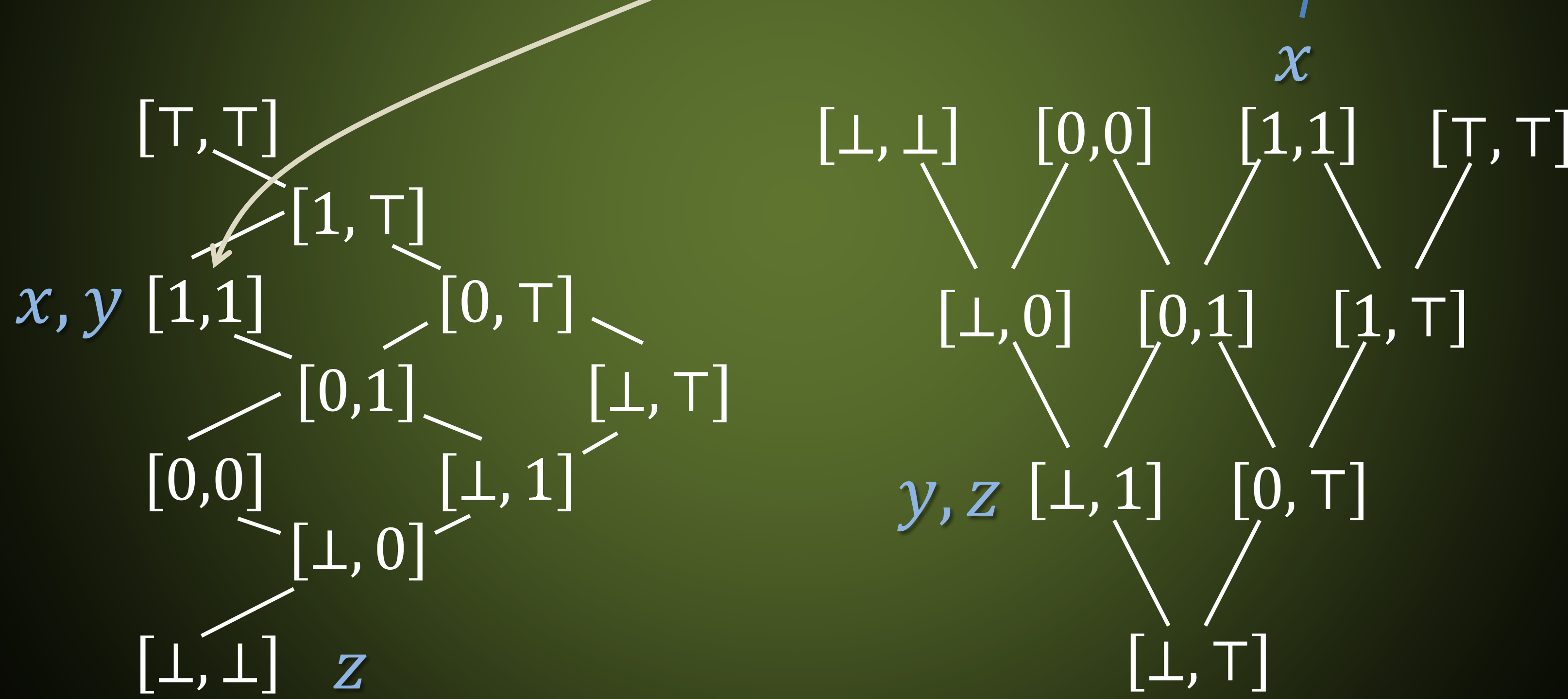


Concurrent

Abstract Analysis

Example

$$\langle\langle ¡x\ ;\ x\ ?\ !y : !z \rangle\rangle_C^S \vee \langle\langle !x \rangle\rangle_C^S$$

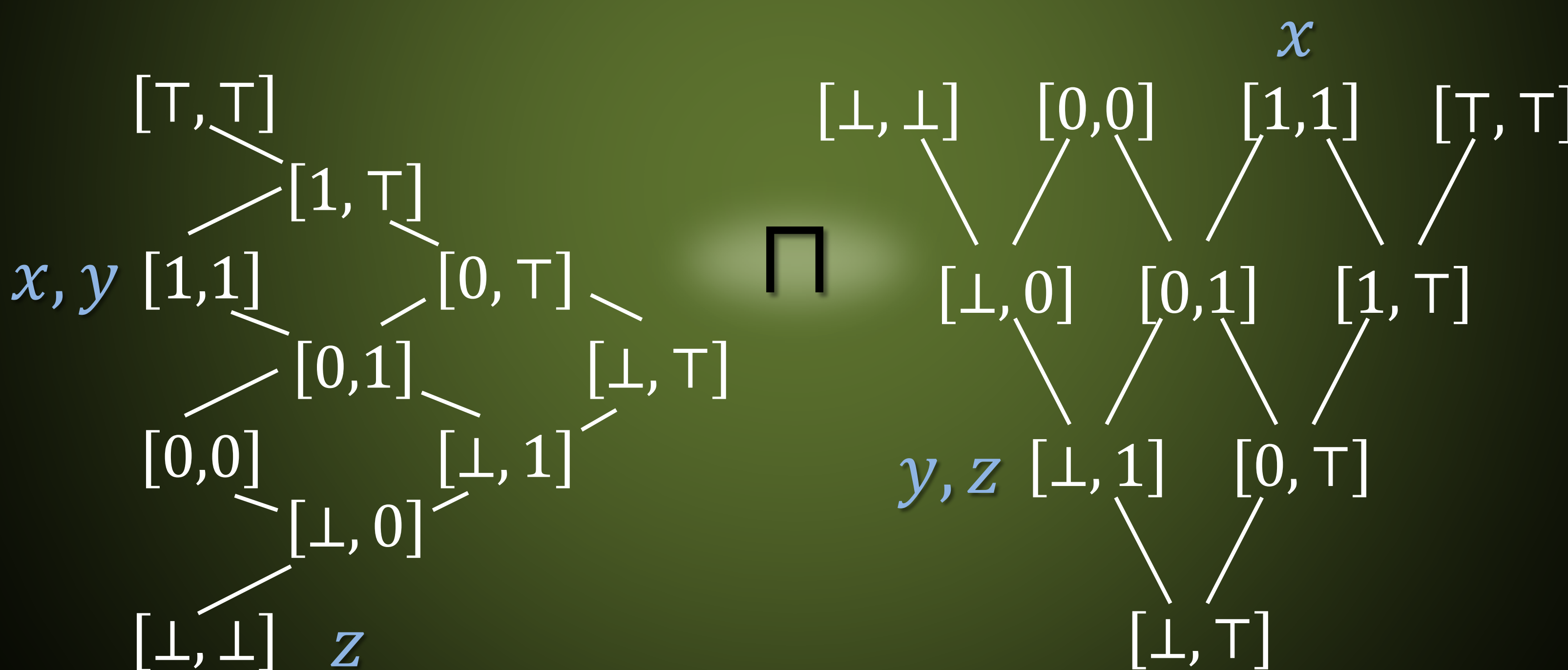


Sequential **Concurrent**

Abstract Analysis

Example

$$\langle\langle ¡x\ ;\ x\ ?\ !y : !z \rangle\rangle_C^S \vee \langle\langle !x \rangle\rangle_C^S$$



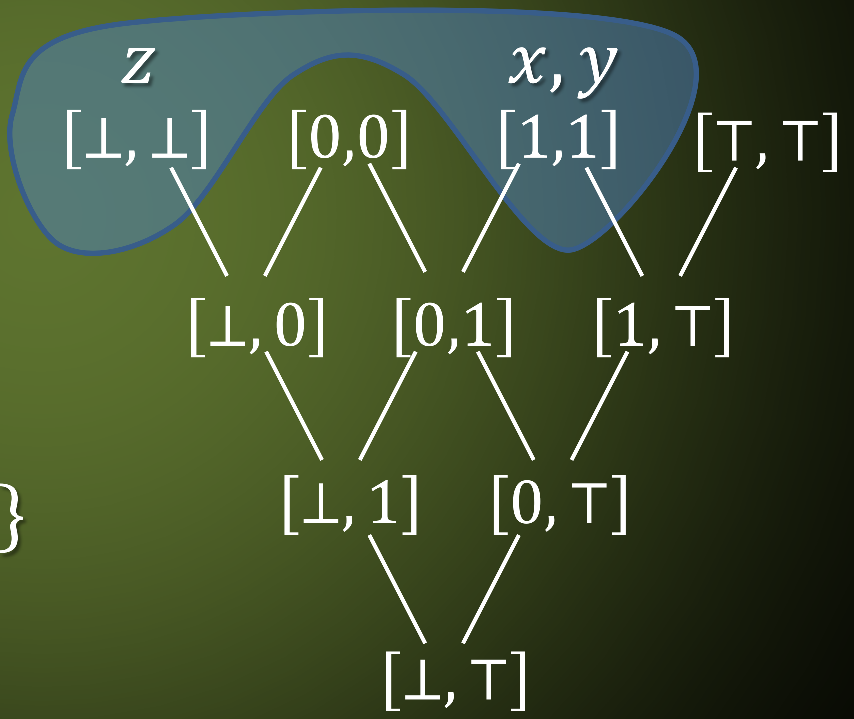
Sequential **Concurrent**

Abstract Analysis

Example

$$\langle\langle ¡x \;;\; x\,?\,!y : !z\rangle\rangle^S_C \vee \langle\langle !x\rangle\rangle^S_C$$

$$\mu C.\langle\langle P\rangle\rangle^S_C = \{\, x : [1,1] \quad y : [1,1] \quad z : [\bot,\bot] \,\}$$

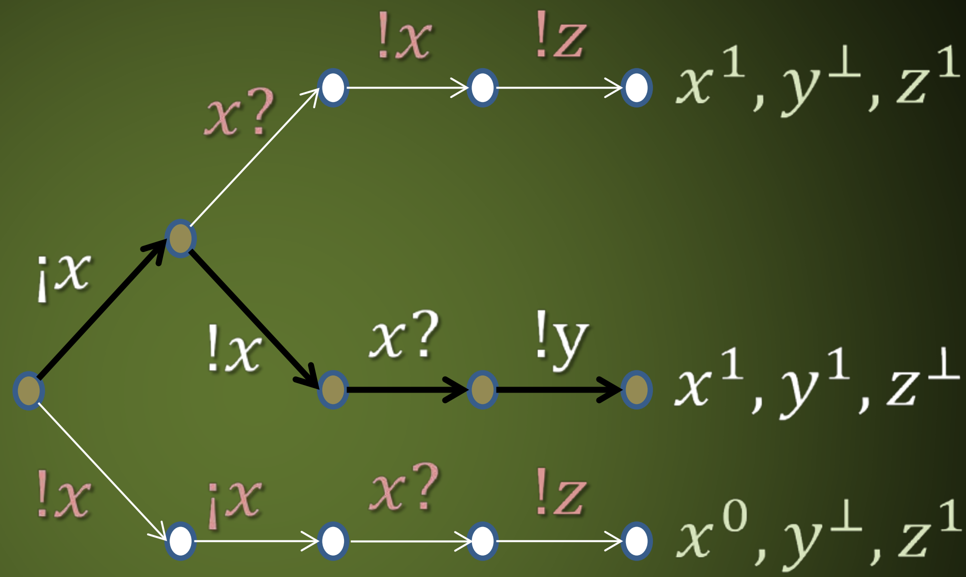


Concurrent

Abstract Analysis

Example

$¡x; x?\,!\,y{:}\,!\,z \,||\, !x$



$\mu C.\langle\langle P\rangle\rangle_C^S$

$x : [1,1]$

$y : [1,1]$

$z : [\perp, \perp]$

Constructiveness Results

Definition:

Program P is:

strongly Berry-constructive (Δ_0-constructive) iff

$$\forall x \in V. (\mu C. \langle\langle P \rangle\rangle_C^{\perp})(x) \in \{\perp, 0,1\}$$

Berry-constructive (Δ_1-constructive) iff

$$\forall x \in V. \left(\mu C. \langle\langle P \rangle\rangle_C^{0}\right)(x) \in \{0,1\}$$

Theorem:

P is Δ_0-constructive implies that P is Δ_1-constructive and
P is Δ_1-constructive implies that P is *SC*.

Conclusions

Signals can be emulated and generalised using share variables + synchronisation constraints.

$\mathcal{SC}$ permits arbitrary (IUR)* tick cycles.

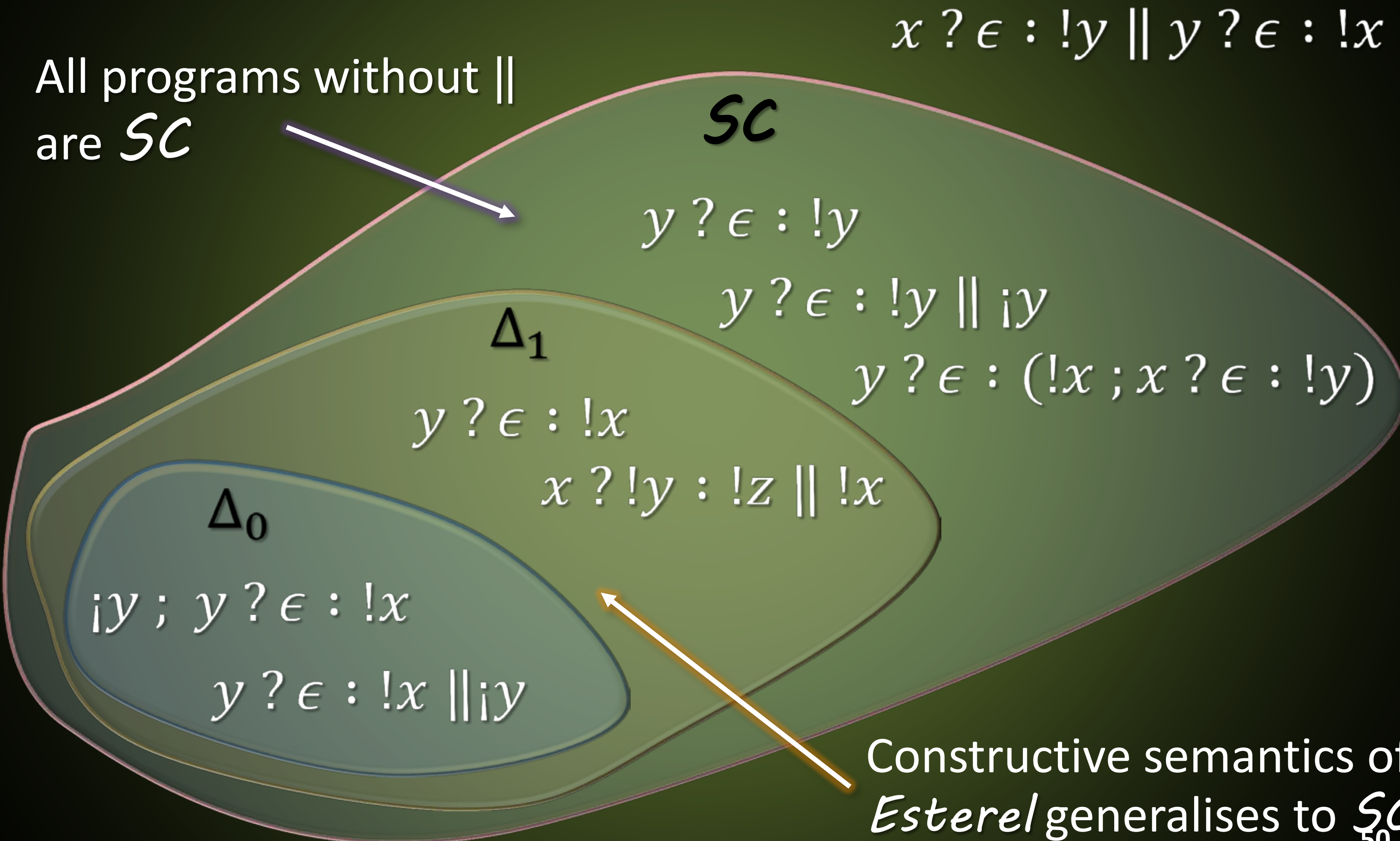
B*e*rry-constructive reactions corresponds to a single (IUR) tick cycle.

Fixed point analysis on sequential\parallel lattice $I(D)$.

$\mathcal{SC}$ is a conservative extension of B*e*rry-constructiveness.

Conclusions

All programs without || are *SC*



Constructive semantics of *Esterel* generalises to *SC*.

Open Problems

Extend results to full *Esterel* (V7) syntax.

Develop fixed point semantics for *SC*.